An Algorithmic Approach to Manifolds

An Analytical Approach to Form Modeling As an Introduction to Computational Morphology

Rémi Barrère

An algorithmic approach to manifolds is presented, based on an object approach to the parametric plotting commands. The initial purpose was to blend geometric and symbolic aspects, so as to equip computer-assisted design (CAD) with symbolic capabilities. Nevertheless, this investigation aims more generally at providing a uniform treatment of analytic geometry and field analysis, in view of applications to physics, system modeling, and morphology.

After presenting the data structure, the core of this article describes a range of operators for manipulating manifolds. It stresses their potential use in shape design and scene description, in particular their ability to supersede several graphics packages. As such, the data type constitutes the foundation of a computational morphology. Then, various extensions are discussed: fields, mesh generation for finite element software, and the prospect of extending the vector analysis package, with emphasis on tensors and differential forms.

■ Introduction

Computer algebra and symbolic programming have introduced analytical capabilities into many areas of scientific computing, such as discrete systems, algebra and summation, calculus, and differential equations. Nevertheless, little benefit has been gained in shape design. Research in that domain has stimulated the evolution of computer-assisted design (CAD), but, so far, these tools have included little or no symbolic capabilities, and most CAD software is still developed with procedural languages and numerical methods. Besides, geometric problems have been tackled so far mainly by means of algebraic or theorem proving methods [1], thus leading to an underdevelopment of analytical methods.

Yet the analytical approach to geometry constitutes the foundation of many physical questions, especially those linked to space or spacetime analysis by means of field theory. Hence, we attempt to introduce analytical capabilities into computer-assisted geometry, initially in view of applications to shape description, then with the purpose of laying the foundation of further uses in differential geometry and field analysis. Although geometry is commonly thought of as the unifying mathematical abstraction underlying those questions, morphology appears as an encompassing common denominator that is able to take into

account questions outside the field of geometry, such as scene description, linkage design, or finite element analysis.

□ Morphology As a Transverse Concept

Although shape and form are more or less considered equivalent in common language, shape pertains to external geometric aspects, whereas form pertains to more general internal structuring aspects (geometric or not).

For instance, curves, surfaces, and volumes as such are shapes, while considered with some other characteristics, such as a field, they tend to be thought of as forms. So a form description is a piece of information about the way in which an object occupies and structures space. In practice, it is inherent to the way the object (or its form) is generated and can be transformed or combined with others.

Moreover, as a result of our cognitive capabilities, we tend to understand objects in contrast to a background and also to perceive them as structured in a container (wrapping structure) with content (internal structure) [2]. On the contrary, modern ideas about space (or spacetime) tend to identify it with its structuring content, space being constituted by the relationships between objects. To some extent, these opposing ideas can be made compatible by blending continuous aspects (figures as expansions) and discrete ones (figures as objects),

□ Mathematical and Algorithmic Requirements

Many form descriptions have been developed so far, in view of more or less specific applications such as drawing software, geometric reasoning, CAD, fractal image generation, and data visualization. There is probably no universal form description, for those methods always need, to some extent, to be optimized to best serve some specific problems. However, the trend to develop ad hoc optimized solutions for practical needs yields "a widely scattered conglomeration of disparate and, at first sight, unrelated methods" [3]. This subsequently tends to severely decrease so-called orthogonality, that is, the capability to cross-fertilize disciplines by information exchange and object combinations thanks to generic types.

By quoting Lord and Wilson [3], we may even stress the need for a mathematical foundation for a science of morphology unifying various approaches, a need that has not been fully recognized yet. Today, the requirement that the proposed method should lend itself well to an algorithmic treatment must be taken into account. It should also have a broader scope than so-called mathematical morphology, which is more or less restricted to the "pixel level" methods used for image processing. In the following, we focus on form synthesis rather than form analysis. The solution put forward derives from a common mathematical tool, slightly adapted to an algorithmic purpose, that is, a new glance at a classical theory [4].

□ Manifolds As a Unifying Approach to Morphology

We focus on manifolds because of their sound analytical foundation, their strong geometric flavor, and their adaptability to algorithmic treatment, especially using computer algebra and symbolic programming. In particular, a manifold determines a codomain as a subset of some (often Euclidean) space. By regarding the codomain as the object and the surrounding space as the background, the manifold actually determines a shape. Various structuring elements, such as a coordinate system (domain) or a field over the manifold, determine as many forms. Moreover, the boundary of a manifold constitutes an element of a container-content approach. More generally, other structuring tools, such as atlases that enable scene descriptions, will be described later.

Because of its visual aspects, shape design is in a natural relationship with the underlying graphics capabilities [5, 6]. In *Mathematica*, these rely upon a few graphics primitives (e.g., `Line`, `Polygon`) and a range of standard plotting commands (e.g., `Plot`, `ParametricPlot`), plus a variety of complementary commands, such as `RevolutionPlot3D` (Version 6) or those in `Graphics‘Shapes‘` (legacy standard packages). This leads to a functional approach where shapes are produced by plotting commands that build them by assembling low-level graphics primitives, thus excluding higher-level objects.

On the contrary, resorting to manifolds leads to a reification of shape: following philosophers, we call reification the mental act of regarding an action as an object. Then, manifolds and other geometric entities like fields can be defined as quite compact symbolic objects, rather than huge assemblies of raw graphics primitives resulting from plotting commands. This facilitates the introduction of both higher-level entities, able to describe objects as wholes, and higher-level symbolic functions able to manipulate and combine these entities. This also enables the consistent gathering of graphics tools scattered around various commands or packages and their extension not only to shape design but also to field analysis.

□ Previous Work

Beyond the classical literature about manifolds [7], Oprea [8] and Gray [9] tackled the question from the computational viewpoint, the former with Maple, the latter with *Mathematica*. More specifically, Tazawa [10, 11] focuses on differential geometry. However, these authors do not introduce any data type for manifolds. Further, a wealth of literature is devoted to surface or solid modeling in view of CAD applications [12], or applications in computational geometry [13]. In most cases, shapes are represented or approximated by splines, Bézier's patches or nonuniform rational B-splines (NURBS) [14], which have become a de facto standard in commercial modeling systems because of their power to represent both free-form shapes and common analytical shapes. Despite its title, [15] only introduces a variant of splines.

An experimental set of packages was developed to investigate the ideas presented in this article. `Morphology‘Master‘` is simply intended to load the whole directory.

```
In[1]:= Needs["Morphology`Master`"]
```

■ An Algorithmic Approach to Manifolds

Although a manifold is usually defined as a collection (called an atlas) of patches, that is, homeomorphisms from an open subset of $\mathbb{R}^n$ onto an open subset of a topological space $\mathcal{M}$ [9], the algorithmic approach will rather focus on $\mathbb{R}^n$ as a curvilinear coordinate system on $\mathcal{M}$, which in most cases will be some subset of $\mathbb{R}^p$. Indeed, the notion of a manifold is deeply rooted in analytical geometry, that is, in the practical need to describe curves, surfaces, and volumes, or their n-dimensional generalizations, both from analytical and geometrical points of view.

□ The Data Structure

As a reflection of this origin, the algorithmic approach aims at blending the analytical and geometric aspects, especially their graphic counterpart. The type `Manifold` introduced hereafter denotes a patch with a specific coordinate system; then an atlas is simply a collection of manifolds, without any continuity or differentiability requirement; so we slightly depart from the usual mathematical definitions.

The type `Manifold` consists of two entities: a set of expressions that should be thought of as a list of parametric equations with a domain specification for the variables that are the local coordinates on the manifold. The type being the head of the expression leads to the informal expression template: `Manifold[`*list of expressions*`,` *domain specification*`]`. The domain specification follows the syntax of continuous domains in plotting commands: a triple or a sequence of triples `{`c`,` $c_{\min}$`,` $c_{\max}$`}` denotes a coordinate symbol with a minimum value and a maximum value. This design is adapted from a first attempt by Gray [16], who actually adopted a more general viewpoint by considering mappings. Here is an example of a segment of a ring with the symbolic parameter `r`.

```
In[2]:= m = Manifold[{ρ Cos[θ], ρ Sin[θ]}, {ρ, 1, r}, {θ, 0, 3 π/2}];
```

In that case, the codomain is a subset of $\mathbb{R}^2$ and the expressions in the first list are usually interpreted as the parametric representation (equations) of a geometric domain. Nevertheless, manifolds describe in principle abstract entities that entail no assumption about their nature, except that they are elements of a topological space. In particular, the parametrized objects need not be points nor vectors. Any parametrized family of entities can be described as a manifold, provided it has some differentiability or at least continuity properties. For instance, a parametrized family of matrices or a parametrized family of functions can be investigated as manifolds. However, there is no standard visualization procedure for such manifolds, and their graphic representation may require assumptions or tricks.

When the codomain is a subset of $\mathbb{R}^p$, it need not be Euclidean; manifold theory is a relativistic theory of abstract spaces. Nevertheless, in many applications, manifolds are interpreted from an absolute point of view: the codomain is supposed to be some subset of a Euclidean space with an orthogonal Cartesian coordinate

system and the domain is viewed as a curvilinear coordinate system for the codomain. Such an absolute interpretation is almost unavoidable when graphic representations are considered.

However, this assumption is not mandatory, and manifolds may have other interpretations: for instance, in another context, the codomain may be a subset of a Euclidean space with a curvilinear coordinate system or even a subset of a space with no metric property. The data type `Manifold` is flexible insofar as there is no strong assumption regarding the absolute or relative interpretation attached to it. In particular, a command like `Embed` is introduced that facilitates switching between the absolute and relative interpretations.

Although it may be required in some contexts, the distinction between open or closed intervals is useless in this algorithmic context; hence, the use of lists for domains. These algorithmic versions of manifolds need not be differentiable, that is, they describe varieties as well (e.g., fractal varieties). Finally, a manifold with no domain is a point. Its syntax is `Manifold[`*list of expressions*`]`. Taking into account this limiting case is useful in some generic applications. When working with manifolds, there is no longer any notion of point, curve, surface, or volume as types since all are manifolds. The number of coordinates gives the nature of the figure, with the dimension of the codomain specifying the embedding space.

□ The Selectors

The selectors extract the various arguments of a typed expression [17]. They were initially introduced in computer science to isolate the interface specification from the internal representation. When the representation is stable, argument extraction can be done directly with patterns in the left-hand sides of transformation rules. Nevertheless, some generic programs are more easily designed by resorting to selectors.

By thinking of applications to geometry or field analysis, we use `Coordinates` for the variables, `Domain` for the coordinate ranges, and `Codomain` for the parametric expressions (thus identifying the functions and their values).

```
In[3]:= Column[Through[{Coordinates, Domain, Codomain}[m]]]

        {ρ, θ}
Out[3]= {{ρ, 1, r}, {θ, 0, 3π/2}}
        {ρ Cos[θ], ρ Sin[θ]}
```

There might be an ambiguity between domain and codomain, which in this context are the domain and codomain of the associated mapping, while the codomain can also be thought of as a domain in a geometric sense. We nevertheless maintain domain and codomain because of their common use in mathematics as well as computer science, and we will resort to patch or region to denote a geometric domain.

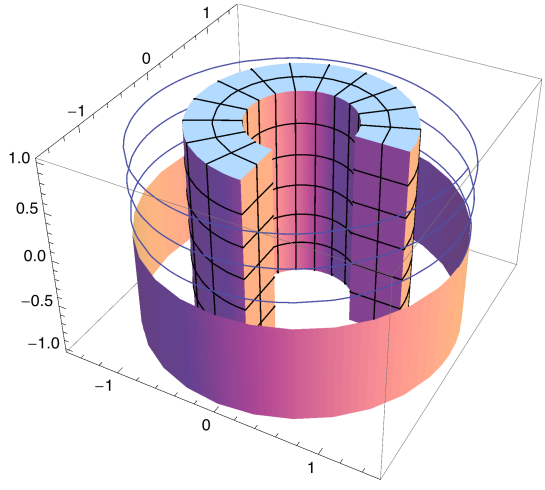
□ Variants

Atlases and Geometric Scenes

In the frame of this algorithmic approach, an atlas is simply a list of manifolds; so we depart from the mathematical approach where a differentiable (or at least continuous) overlapping is required. Here, an atlas is a piecewise manifold; it is slightly more general for it enables the accumulation of manifolds with different dimensions. The manifolds of an atlas must nonetheless have codomains with the same dimensions.

```
In[4]:= Draw[Atlas[{
          Manifold[Cylindrical[{r, θ, h}],
           {r, 1/2, 1}, {θ, 0, 3 π/2}, {h, -1, 1}],
          Manifold[Cylindrical[{3/2, θ, h}], {θ, 0, 2 π}, {h, -1, 0}],
          Manifold[Cylindrical[{3/2, t, t/6/π}], {t, 0, 6 π}]
         }], Mesh → {{1, 12, 5}, None, Automatic}]
```

Out[4]=



Combined with `Draw`, an `Atlas` can be viewed as an adaptation to manifolds of the command `StackGraphics` from the former package `Graphics`Graphics3D`` (legacy standard packages) that directly applies to graphics objects.

Atlases are especially useful for manipulating sets of manifolds as wholes. They constitute a natural data structure for compound shapes or geometric scenes, the atlas then being thought of as a set of figures. Manifolds then appear as the natural primitives for scene description, in association with the type `Atlas` as a composition tool. As such, they enable combining continuous and discrete aspects.

Many-Valued Manifolds

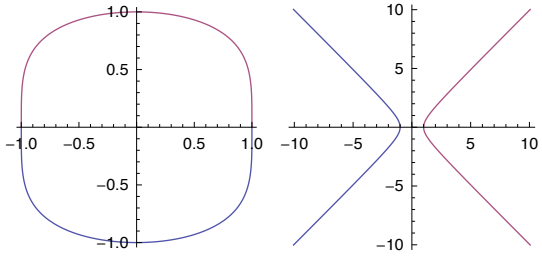
A many-valued manifold differs from an atlas in the sense that there is a common domain for a list of parametrizations; nevertheless, it can possibly be rewritten as an atlas of manifolds. The multiple parametrizations cannot only be arranged in lists but also in matrices or higher-dimensional tables. Roughly, many-valued manifolds implement coverings. These are mainly used to parametrize figures with several branches (e.g., a hyperbola), in particular in the case of manifolds computed as inverse mappings. Many-valued manifolds also allow an application to fractals (see the section Manifolds and Fractals).

```
In[5]:= Solve[x^2 + y^4 == 1, y]
```

$$\text{Out[5]}= \left\{\left\{y \to -\left(1-x^2\right)^{1/4}\right\}, \left\{y \to -i\left(1-x^2\right)^{1/4}\right\}, \left\{y \to i\left(1-x^2\right)^{1/4}\right\}, \left\{y \to \left(1-x^2\right)^{1/4}\right\}\right\}$$

```
In[6]:= Draw[{Manifold[{x, y} /. Drop[%, {2, 3}], {x, -1, 1}],
          Manifold[{{-Cosh[φ], Sinh[φ]}, {Cosh[φ], Sinh[φ]}}, {φ, -3, +3}]}]
```

Out[6]=



Coordinate Systems and Manifolds

In view of a generic treatment of fields, coordinate systems should be regarded as particular cases of coordinate systems on manifolds. This is achieved by focusing on the transformations associated with the coordinate systems; for example, `Polar[{r, θ}]` refers to the polar coordinate system. There is nonetheless no default symbol for coordinates: freely chosen by the user, these are specified where needed.

```
In[7]:= Polar[{r, θ}]
```

```
Out[7]= {r Cos[θ], r Sin[θ]}
```

The associated rewrite rules must be given for every common coordinate system. A symbol like `Polar` may express either a change of coordinates (relative viewpoint) when processed as a function, or a coordinate system (absolute viewpoint) when processed as a type (symbol wrapping a list of coordinates). For that purpose, the argument of `CoordinateSystem` is held with the appropriate attribute.

```
In[8]:= Polar[{r, θ}] // CoordinateSystem
```

```
Out[8]= CoordinateSystem[Polar[{r, θ}]]
```

Although it might seem counterintuitive at first sight, the relative viewpoint turns out to be more practical and more meaningful, too: the idea of an absolute coordinate system has no serious foundation and is conventional in the end. In

particular, `Manifold[Polar[{r, θ}]]` yields `Manifold[{r Cos[θ], r Sin[θ]}]`, that is, the Cartesian coordinates of the point. So it is equivalent to the command `CoordinatesToCartesian` of the vector analysis package, the analog of `CoordinatesFromCartesian` being obtained by introducing reciprocal coordinate systems (respectively, coordinate changes), for example, `InversePolar[{u, v}]`.

```
In[9]:= {InverseFunction[Polar], InversePolar[{u, v}]}

Out[9]= {InversePolar, {Sqrt[u^2 + v^2], ArcTan[u, v]}}
```

With these conventions, the expression `Manifold[Polar[{f[r, θ], g[r, θ]}], {r, r_min, r_max}, {θ, θ_min, θ_max}]` describes a sector of a ring and `Manifold[Polar[{f[t], g[t]}], {t, t_min, t_max}]` describes a curve with parametric expressions `f[t]` and `g[t]` in polar coordinates. Curves and surfaces in three-dimensional coordinate systems can be given similarly. Although it is uncommon, this use of parametric representations in some coordinate system turns out to be a concise and convenient way to describe manifolds.

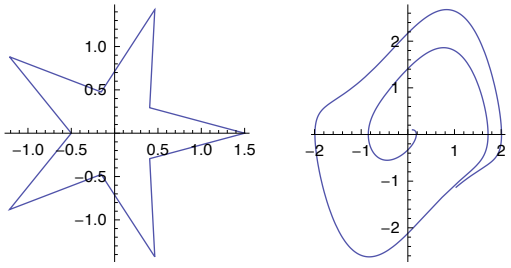
Manifolds with Interpolating Functions

If need be, a manifold can be defined in terms of interpolating functions. Convenient when solving differential equations (`NDSolve`), such manifolds are occasionally useful for building more or less intricate shapes. Splines, Bézier manifolds, or NURBS could also be used [18].

```
In[10]:= theStar = {Interpolation[Table[{t, (1 + Cos[5 t] / 2) Cos[t]},
             {t, 0., 2 π, π / 5}], InterpolationOrder → 1],
           Interpolation[Table[{t, (1 + Cos[5 t] / 2) Sin[t]},
             {t, 0., 2 π, π / 5}], InterpolationOrder → 1]}

Out[10]= {InterpolatingFunction[{{0., 6.28319}}, <>],
          InterpolatingFunction[{{0., 6.28319}}, <>]}

In[11]:= vanDerPol[t_] = {y[t], z[t]} /.
           First[NDSolve[{y'[t] == z[t], z'[t] == (1 - y[t]^2) z[t] - y[t],
              y[0] == 0.1, z[0] == 0.1}, {y, z}, {t, 0, 15}]]

Out[11]= {InterpolatingFunction[{{0., 15.}}, <>][t],
          InterpolatingFunction[{{0., 15.}}, <>][t]}
```

```
In[12]:= Draw[{Manifold[Through[theStar[t]], {t, 0, 2 π}],
           Manifold[vanDerPol[t], {t, 0, 15}]}, AspectRatio → Automatic]
```

Out[12]=



□ Mappings

Mappings constitute a powerful tool that implement not only geometric transformations but also changes of coordinates. In our algorithmic context of typed entities, there are two levels: vector functions that apply to n-tuples (lists) and mappings that apply to manifolds. Vector functions are defined according to the informal scheme `someTransform[{s1_, s2_, …}] := {expressions of the si}`. For instance:

```
In[13]:= nonLinear[{x_, y_}] := {x - 3 y, y + x^2}
```

In the following, we will have to manipulate such vector functions or even lists of vector functions; hence, the usefulness of a construct for pure vector functions.

Pure Vector Functions and Function Systems

A pure vector function is a variant of pure functions with a single argument, which is a list plus a means to refer to the list elements individually in the definition. Then, such a vector function can be used in place of a definition like `nonLinear`. In the case of nested vector functions, unique symbols are introduced.

```
In[14]:= VectorFunction[{x, y}, {x - 3 y, y + x^2}]

Out[14]= {#1[[1]] - 3 #1[[2]], #1[[1]]^2 + #1[[2]]} &

In[15]:= nonLinear[{u, v}] == %[{u, v}]

Out[15]= True
```

By using `Through`, the construct `FunctionSystem` distributes the functions (or vector functions) of a list over the argument(s).

```
In[16]:= FunctionSystem[{ArcTan, #1 * #2 + 1 &}][u, v]

Out[16]= {ArcTan[u, v], 1 + u v}

In[17]:= FunctionSystem[
           {nonLinear, VectorFunction[{x, y}, {x + y, x y}]}][{u, v}]

Out[17]= {{u - 3 v, u^2 + v}, {u + v, u v}}
```

Modularity prevents the use of such vector functions in the case of manifolds.

Moreover, these do not allow the useful notion of parametrized transformation; hence, the introduction of mappings.

Mappings and Parametrized Mappings

Mappings constitute one of the most fundamental operations on manifolds. Many others can be defined in terms of mappings. A mapping is a parametrized transformation (i.e., a continuous set of transformations) with a domain specification for each parameter that applies to manifolds and yields a manifold. When there is no parameter, mappings amount to ordinary manifold transformations; otherwise, they implement extrusions as shown in the section Operations on Manifolds. Mappings are defined as subvalues. Their syntax is similar to that of manifolds, except that the first term is a vector function or a list of vector functions in the case of multiple mappings: `Mapping[`*transformation*`,` *domain*`]`.

```
In[18]:= Mapping[tr, {c, 0, Pi}]@
          Manifold[{f[u, v], g[u, v]}, {u, 0, 1}, {v, a, b}]
```

```
Out[18]= Manifold[tr[{f[u, v], g[u, v]}], {u, 0, 1}, {v, a, b}, {c, 0, π}]
```

Although they are similar, mappings differ from vector functions: the latter apply to vectors, that is, lists, while the former apply to manifolds. A mapping is a construct that applies transformations to manifolds. In the following, the rotation operator applies only to lists. The wrapper `Mapping` enables its application to manifolds and the introduction of parameters. This approach enables modularity since transformations need no longer be explicitly defined for manifolds.

```
In[19]:= Rotation[2 π / 3][{x, y}]
```

$$\text{Out[19]= } \left\{-\frac{x}{2}-\frac{\sqrt{3}\,y}{2},\ \frac{\sqrt{3}\,x}{2}-\frac{y}{2}\right\}$$

```
In[20]:= Rotation[2 π / 3][Manifold[{x, y}, x, y]]
```

$$\text{Out[20]= } \text{Rotation}\left[\frac{2\pi}{3}\right][\text{Manifold}[\{x, y\}, x, y]]$$

```
In[21]:= Mapping[Rotation[2 π / 3]][Manifold[{x, y}, x, y]]
```

$$\text{Out[21]= } \text{Manifold}\left[\left\{-\frac{x}{2}-\frac{\sqrt{3}\,y}{2},\ \frac{\sqrt{3}\,x}{2}-\frac{y}{2}\right\}, x, y\right]$$

```
In[22]:= Mapping[Rotation[a], {a, 0, π}][Manifold[{t, 0}, {t, 0, 1}]]
```

```
Out[22]= Manifold[{t Cos[a], t Sin[a]}, {t, 0, 1}, {a, 0, π}]
```

Like a pure function, a mapping is an inert object that becomes active when applied to a manifold. The relationship between manifolds and mappings is similar to that between expressions and functions. A manifold can always be thought of as some mapping applied to a generic Cartesian manifold. Here is another standard decomposition with a parametrized mapping applied to the origin.

```
In[23]:= {Manifold[tr[{x, y, z}], d] == Mapping[tr]@Manifold[{x, y, z}, d],
          Manifold[{u[x], v[y]}, d] ==
           Mapping[{u[x], v[y]} + # &, d]@Manifold[{0, 0}]}

Out[23]= {True, True}
```

Multiple Mappings and Sheaves

Multiple mappings are intended to apply systems of functions to manifolds. A multiple mapping generates a many-valued manifold according to the scheme

```
In[24]:= Mapping[{tr1, tr2}, {c, 0, Pi}]@
          Manifold[{f[u, v], g[u, v]}, {w, 0, 1}, {v, a, b}]

Out[24]= Manifold[{tr1[{f[u, v], g[u, v]}], tr2[{f[u, v], g[u, v]}]},
          {w, 0, 1}, {v, a, b}, {c, 0, π}]
```

Multiple mappings are useful for duplicating manifolds or implementing iterated function systems in terms of manifolds (see the section Manifolds and Fractals). They are nonetheless restricted to the case when the transformations share the same parameter domain (or have no parameter). In the general case, multiple mappings require a specific construct: sheaves. Sheaves are collections of mappings. Like multiple mappings, they are intended to apply function systems to manifolds. A sheaf generates an atlas according to the scheme

```
In[25]:= Sheaf[{Mapping[tr1, d1], Mapping[tr2, d2]}][Manifold[{pr}, coord]]

Out[25]= Atlas[
          {Manifold[tr1[{pr}], coord, d1], Manifold[tr2[{pr}], coord, d2]}]
```

■ Visualization of Manifolds

The visualization of manifolds mainly relies upon the `ParametricPlot` and `ParametricPlot3D` commands, plus the novel function `SolidParametric`. `Plot3D` developed for three-dimensional shapes. A generic command is introduced in the form of polymorphic transformation rules.

□ The Command Draw

Since both the functions to be plotted and the domains are part of the data structure, manifolds require a single graphic command with a single argument plus possible options; let us call it `Draw`. Only manifolds with a codomain that has a geometric interpretation in a two- or three-dimensional space are drawn; the length of the first argument determines the dimension of the representation space. Then the number of coordinates determines the nature of the represented object: point, curve, surface, volume, or higher-dimensional object. Moreover, the codomain is supposed to be a subset of a Euclidean affine space equipped with a canonical orthogonal Cartesian coordinate system, which has consequently the status of an "absolute" reference space. Then the domain is that of the curvilinear coordinate system of the manifold.

Most drawings are "subcontracted" to standard parametric plotting commands.

n (domain)	p (codomain)	Plotting Function
0	2	`Graphics`
0	3	`Graphics3D`
1	2	`ParametricPlot`
1	3	`ParametricPlot3D`
2	2	`ParametricPlot`
2	3	`ParametricPlot3D`
3	3	`SolidParametricPlot3D`

Table 1. A summary of the various combinations.

An extension of the `ParametricPlot` family is required for the 3×3 case. It has been temporarily called `SolidParametricPlot3D`, although polymorphism should allow processing it as a specific case of `ParametricPlot3D`. `SolidParametricPlot3D` draws the boundary, which is valid only when the Jacobian of the parametric equations does not vanish. The case $p = 1$ is not processed by `Draw` because of its weak visual interest. Nevertheless, thanks to a lifting, it can be rearranged into a two-dimensional or three-dimensional manifold so as to be visualized the same way functions are visualized with `Plot` or `Plot3D`.

```
In[26]:= Lift[Manifold[{h[t]}, {t, -1, 1}]]

Out[26]= Manifold[{t, h[t]}, {t, -1, 1}]
```

□ Higher-Dimensional Manifolds

A higher-dimensional manifold is processed by means of the two-dimensional or three-dimensional projection of its boundary. Here is an example of the four-dimensional unit hypercube [19].

```
In[27]:= unitHyperCube4D = UnitHyperCube[{a, b, c, d}]

Out[27]= Manifold[{a, b, c, d}, {a, 0, 1}, {b, 0, 1}, {c, 0, 1}, {d, 0, 1}]
```

In the following, a three-dimensional projection of the four-dimensional unit hypercube is drawn. The transparency effects of Version 6 are appreciated. Analog applications to complex-valued functions are also possible [20].

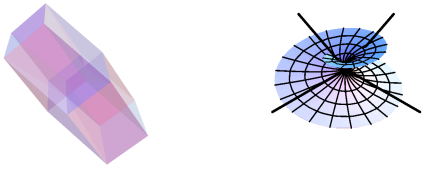
```
In[28]:= polarRiemannSurface4D[fz_,
           {r_, rmin_, rmax_}, {θ_, θmin_, θmax_}] := Manifold[
           {r Cos[θ], r Sin[θ], ComplexExpand[Re[fz /. z → r Exp[i θ]]],
            ComplexExpand[Im[fz /. z → r Exp[i θ]]]},
           {r, rmin, rmax}, {θ, θmin, θmax}]
```

```
In[29]:= sqrtPolarRiemannSurface4D =
          MapAt[PowerExpand, polarRiemannSurface4D[z^(1/2),
            {r, 0, 3}, {θ, 0, 4 π}] /. Arg[Exp[i θ] r] → θ, 1]
```

$$\text{Out[29]= Manifold}\left[\left\{r\,\text{Cos}[\theta],\ r\,\text{Sin}[\theta],\ \sqrt{r}\,\text{Cos}\left[\frac{\theta}{2}\right],\ \sqrt{r}\,\text{Sin}\left[\frac{\theta}{2}\right]\right\},\ \{r, 0, 3\}, \{\theta, 0, 4\pi\}\right]$$

```
In[30]:= Draw[{Sides[
           Mapping[Shadowing[{{1, -1, 0, 0}, {1, 0, -1, 0}, {1, 0, 0, -1}},
             {1, 2, 1, 2}]]@unitHyperCube4D, 2],
           Mapping[Shadowing[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1/2, 1/2}},
             {1, -1, 1, -1}]]@Atlas[{Manifold[
              {{1.5, 0, 0, 0}, {0, 1.5, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}} t,
              {t, 0, 4}], sqrtPolarRiemannSurface4D}]
         }, Axes → None, Boxed → False, PlotPoints → {2, {2, {5, 40}}},
         Mesh → {None, {None, {3, 35}}},
         PlotStyle → {Opacity[0.3], Thickness[0.01], {}},
         ViewPoint → {{{0.9, -2.5, 2.1}}, {{1.9, 1.9, 1.9}}}]
```

Out[30]=



The remainder of this article describes a variety of tools that extend the game by enabling numerous manifold manipulations.

■ Shapes as Manifolds

Thanks to their graphic representations, manifolds can be used for shape description and visualization. As such, they constitute a means for introducing symbolic aspects in shape design. This relies upon a number of primitive manifolds and shapes, such as cylinder, sphere, or torus, as well as manifold and shape generators that compute boundaries, embeddings, extrusions, and so on, or that operate by combining or transforming them.

□ Primitive Manifolds and Shapes

A manifold is obtained by combining a vector function (a shape function) and a domain specification in the form of the construct `Manifold[`*vector expression*`,` *domain*`]`. So modularity is simply obtained by defining (hence, naming) a number of vector functions that can then be combined at will with all kinds of domain specifications to produce as many manifolds.

The most common shape functions are related to the usual coordinate systems (e.g., `Polar`, `Cylindrical`, and `Spherical`) that describe objects with polar, cylindrical, or spherical symmetries. These are the ingredients for building circles, disks, rings, cylinders, and spheres. Other shape functions are drawn from classical parametric equations that define tori, Lissajous curves, families of cycloids, quadrics, and so forth. These two families of shape functions are respectively introduced in the packages `Morphology‘Coordinates‘` (automatically loaded by `Morphology‘Manifolds‘`) and `Morphology‘Shapes‘`.

When these shape functions are parametrized, subvalues are used for their definitions. For instance, in the following definition of a parallelogram, `u` and `v` denote two vectors while λ and μ denote the coordinates, that is, the shape function is `Oblique[{{u1, u2}, {v1, v2}}]`.

```
In[31]:= Oblique[{{u1, u2}, {v1, v2}}][{λ, μ}]
```

$$\text{Out[31]=}\ \{\lambda\, u_1 + \mu\, u_2,\ \lambda\, v_1 + \mu\, v_2\}$$

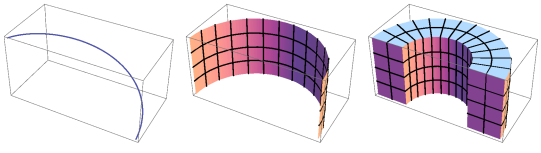
```
In[32]:= Manifold[Oblique[{{1, 1/2}, {0, 2}}][{λ, μ}], {λ, 0, 1}, {μ, 0, 1}]
```

$$\text{Out[32]=}\ \text{Manifold}\left[\left\{\lambda + \frac{\mu}{2},\ 2\,\mu\right\},\ \{\lambda,\ 0,\ 1\},\ \{\mu,\ 0,\ 1\}\right]$$

This way of substituting shape functions for explicit primitive manifolds avoids possible conflicts with the standard graphics primitives and fits the polysemy of the language of geometry, the same words (e.g., ellipse, cylinder, or cone) denoting both lines and surfaces (respectively, surfaces and volumes). Indeed, the domain configuration determines the dimension whatever the shape function: a single list denotes a curve (single coordinate), two lists a surface (two coordinates), and three lists a volume.

```
In[33]:= m = {Manifold[Cylindrical[{r0, π h , h}], {h, 0, 1}],
              Manifold[Cylindrical[{r0, θ, h}], {θ, 0, π}, {h, 0, 1}],
              Manifold[Cylindrical[{r, θ, h}],
               {r, 1/2, 1}, {θ, 0, π}, {h, 0, 1}]};

In[34]:= Draw[m /. r0 → 1, Ticks → None, Axes → None,
          Mesh → {Automatic, {12, 3}, {1, 12, 3}}]
```

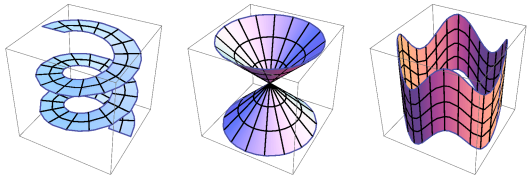
Out[34]=



The same cylindrical shape function generates many other figures, such as a helicoid, a cone, or more exotic figures.

```
In[35]:= m = {Manifold[
             Cylindrical[{r, θ, θ/(2 π)}], {r, 1/2, 1}, {θ, 0, 5 π}],
           Manifold[Cylindrical[{r, θ, r}], {r, -1, 1}, {θ, 0, 2 π}],
           Manifold[
            Cylindrical[{1, θ, h + Sin[4 θ]/2}], {θ, 0, 2 π}, {h, -1, 1}]};

In[36]:= Draw[m, Ticks → None, Axes → None, BoxRatios → {{1, 1, 1}},
          BoundaryStyle → Automatic, Mesh → {{1, 35}, {5, 15}, {25, 3}}]
```

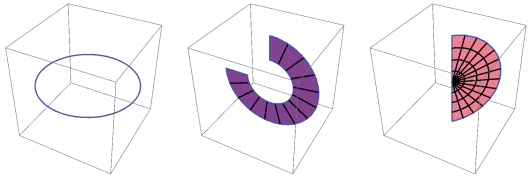
Out[36]=



This approach is especially useful for embedding figures in higher-dimensional spaces without resorting to the embedding operator. Here are the embeddings of a circle, a ring, and half a disk.

```
In[37]:= m = {Manifold[Cylindrical[{1, θ, 0}], {θ, 0, 2 π}],
           Manifold[RotateLeft[Cylindrical[{r, θ, 0}]],
            {r, 1/2, 1}, {θ, 0, 3 π/2}],
           Manifold[Spherical[{r, π/4, phi}],
            {r, 0, 1}, {phi, -π/2, π/2}]};

In[38]:= Draw[m, Axes → None, BoundaryStyle → Automatic,
          Mesh → {Automatic, {0, 12}, {3, 12}},
          PlotRange → {{{-1, 1}, {-1, 1}, {-1, 1}}}]
```

Out[38]=



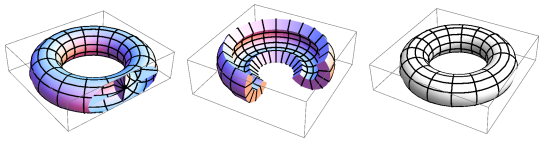
Then, if need be, the shapes can be moved or deformed by means of affine or more general transformations that are introduced next.

□ A Small Gallery of Manifolds

With the drawing command and the novel capacities of Version 6, these primitive manifolds supersede the former `Shapes`` package, including wireframes: here are a few examples drawn from the package. Other examples can be found in the literature, for example, [9] or [21].

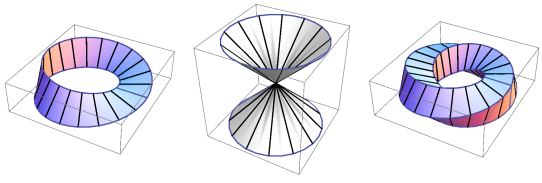
```
In[39]:= Draw[
          {Manifold[Annular[3][{ρ, θ, φ}], {ρ, 0, 1}, {θ, 0, 2 π}, {φ, 0, 2 π}],
           Manifold[Annular[3][{ρ, θ, φ}], {ρ, 1, 2},
            {θ, 0, 3 π/2}, {φ, -3 π/4, π/4}],
           Manifold[Annular[3][{1, θ, φ}], {θ, 0, 2 π}, {φ, 0, 2 π}]},
          {{Mesh → {0, 15, 10}, PlotRange → {{-4, 3}, {-4, 4}, {-1, 1}},
            ViewPoint → {2., -1.5, 1.7}}, {Mesh → {0, 15, 5}},
           {Mesh → {15, 10}, ColorFunction → (White &)}}, Axes → None]
```

Out[39]=



```
In[40]:= Draw[{ Manifold[Moebius[3][{ρ, θ}], {ρ, -1, 1}, {θ, 0, 2 π}],
            Manifold[Cylindrical[{r, θ, r}], {r, -1, 1}, {θ, 0, 2 π}],
            Manifold[Moebius[3][{r, θ, h}],
             {r, -1, 1}, {θ, 0, 2 π}, {h, -1, 1}]
           }, Axes → None, BoundaryStyle → Automatic,
           ColorFunction → {Automatic, (White &), Automatic},
           PlotPoints → {{2, 22}, {2, 14}, {2, 22, 2}},
           Mesh → {{0, 20}, {0, 12}, {0, 20, 0}}]
```

Out[40]=



■ Manifolds and Fractals

□ Manifolds and Fractal Functions

Fractal functions generalize the process by which the Weierstrass function is built. Fractal curves (or, more generally, fractal varieties) can be built by means of fractal functions, obtained by summing scale transformed versions of an initial function, typically a trigonometric function [22].

```
In[41]:= ScaleTransform[w, δ, k][Sin[t], t]
```

Out[41]= $w^{-k\,\delta}\,\text{Sin}\left[t\,w^{k}\right]$

The command `FractalFunction` computes the n^{th} approximation.

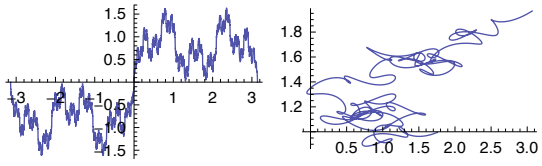
```
FractalFunction[ω_,δ_,n_][e_,t_]:=
    Sum[ScaleTransform[ω,δ,k][e,t],{k,0,n}]
```

Fractal functions can be used to generate parametric approximations to fractal

manifolds. Even when not useful for scientific purposes, they may have aesthetically pleasing properties.

```
In[42]:= Draw[{
           Manifold[{t, FractalFunction[3, 0.5, 7][Sin[t], t]}, {t, -π, π}],
           Manifold[FractalFunction[{3, 4}, {0.3, 0.5}, 5][
             {Cos[t/9], Cos[t/25]}, t], {t, 0, 2 π}]}, AspectRatio → 0.6]
```

Out[42]=



□ Manifolds and Iterated Function Systems

Iterated function systems (IFS) constitute another well-known way to generate fractals [23]. Briefly, an IFS is a set of contracting affine transformations that are iteratively applied to any initial figure. Multiple mappings directly implement the iteration [24] rather than its variant, "chaos game" [24, 25]: they compute the n^{th} approximation of an IFS, from any initial manifold, in the form of a many-valued manifold.

```
In[43]:= Mapping[IFS["Cantor"]]@Manifold[{t, 0}, {t, 0, 1}]
```

$$\text{Out[43]= Manifold}\left[\left\{\left\{\frac{1}{3}+\frac{t}{3},\ \frac{1}{3}\right\},\ \left\{-\frac{1}{3}+\frac{t}{3},\ -\frac{1}{3}\right\}\right\},\ \{t,\ 0,\ 1\}\right]$$

A few common IFSs are defined in the package `Morphology`Fractals``, expressed in the form `IFS["`*Name*`"]` that yields a function system.

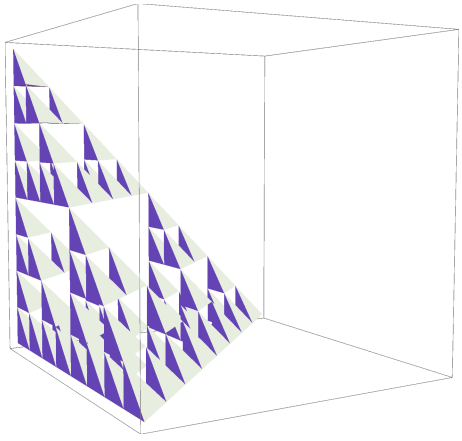
```
In[44]:= IFS["Cantor"]
```

$$\text{Out[44]= FunctionSystem}\left[\left\{\frac{\#1}{3}+\frac{1}{3}\&,\ \frac{\#1}{3}-\frac{1}{3}\&\right\}\right]$$

Here is an example of a Sierpinski sponge.

In[45]:=
```
Draw[Nest[Mapping[IFS["RectangularSierpinski3D"]], Manifold[
   {u, (1 - u) v, (1 - u) (1 - v) w}, {u, 0, 1}, {v, 0, 1}, {w, 0, 1}], 3],
 PlotPoints → 2, Mesh → None, Axes → None,
 ViewPoint → {2.4, -1.4, 0.2}]
```

Out[45]=



Giving a low value to `PlotPoints` is necessary to avoid a lengthy computation. The process appears to be a quite general one that builds a variety where even continuity properties are abandoned.

In association with a convenient initial manifold, an IFS can also produce a branching figure. Such figures are commonly generated with L-systems. Weighted function systems are also possible, which allow visualizing invariant measures on fractals by means of the chaos game. Here is an example of a Sierpinski triangle with weights 1 / 2, 1 / 4, and 1 / 4.

In[46]:=
```
branch[1][v_] := v / 2
branch[2][v_] := v / 2 + {0, 1 / 2}
branch[3][v_] := Rotation[Pi / 6, 0][v / 3] + {0, 1 / 3}
branch[4][v_] := Rotation[-Pi / 6, 0][v / 3] + 2 {0, 1 / 3}
```

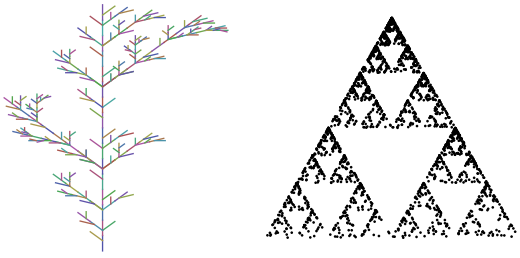
In[50]:=
```
Nest[Mapping[Array[branch, 4]], Manifold[{0, t}, {t, 0, 1}], 1]
```

Out[50]=

$$\text{Manifold}\left[\left\{\left\{0, \frac{t}{2}\right\}, \left\{0, \frac{1}{2}+\frac{t}{2}\right\}, \left\{-\frac{t}{6}, \frac{1}{3}+\frac{t}{2\sqrt{3}}\right\}, \left\{\frac{t}{6}, \frac{2}{3}+\frac{t}{2\sqrt{3}}\right\}\right\}, \{t, 0, 1\}\right]$$

```
In[51]:= Draw[{
           Nest[Mapping[Array[branch, 4]],
            Manifold[{0., 1. t}, {t, 0, 1}], 4],
           Manifold[PlayChaosGame[WeightedFunctionSystem[
              IFS["EquilateralSierpinski2D"],
              {0.5, 0.25, 0.25}], {0., 0.}, 2500]]
         }, Axes → None, PlotPoints → 2, PlotStyle → PointSize[0.005],
         AspectRatio → {1.1, Automatic}]
```

Out[51]=



■ Operations on Manifolds

Beyond the aforementioned primitives, new manifolds or shapes are generated by combining or transforming them.

□ Sides and Boundary

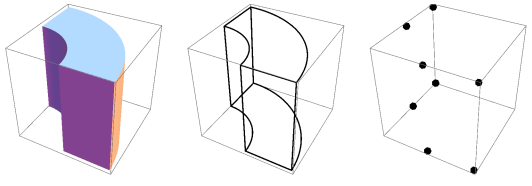
In the two-dimensional case, the first-order sides are edges while the second-order ones are vertices. In the three-dimensional case, the first-order sides are faces, second-order ones are edges, and third-order ones are vertices, and so forth. The first-order sides constitute the boundary from the domain viewpoint: in some pathological cases, the first-order sides are not necessarily the same as the geometrical boundary, as, for example, when the codomain overlaps itself.

```
In[52]:= b0 = Manifold[{r Cos[t], r Sin[t], h},
            {r, 1, 2}, {t, 0, Pi/2}, {h, -1, 1}];
         b1 = Sides[b0]; b2 = Sides[b0, 2]; b3 = Sides[b0, 3]; Short[b2, 1]
```

```
Out[53]//Short= Atlas[{Manifold[{1, 0, h}, {h, -1, 1}],
                  <<10>>, Manifold[{0, r, 1}, {r, 1, 2}]}]
```

```
In[54]:= Draw[{b1, b2, b3}, Mesh → None, Ticks → None,
          Axes → None, PlotStyle → PointSize[0.05]]
```

Out[54]=



□ Left Compositions

Most operations are based on various combinations of manifolds and mappings or parametrized mappings. According to their order, we get the left compositions and their variants that implement transformations, extrusions, and duplications (following sections), or the right compositions and their variants that implement embeddings, connections, changes of coordinates, and reshapings (subsequent sections).

Transformations

In a geometrical context where manifolds are intended to describe figures or shapes, transformations constitute a useful manifold generator. Let us recall that these are defined as vector functions or pure vector functions that apply to *n*-tuples (lists) of coordinates; then the `Mapping` construct enables their application to manifolds.

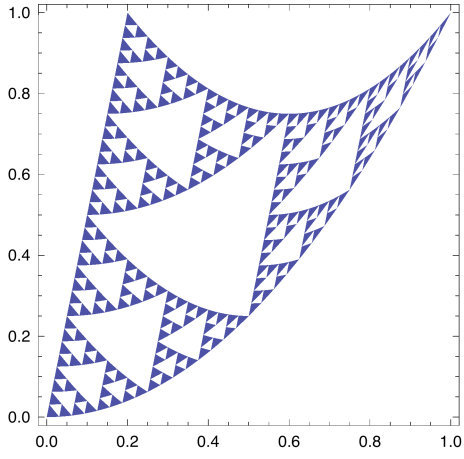
```
In[55]:= deform[{x_, y_}] := {x + y/5, y + x^2}
         Mapping[deform]@Manifold[{u, v}, {u, 0, 1}, {v, 0, 1}]
```

$$\text{Out[56]= Manifold}\left[\left\{u+\frac{v}{5},\ u^2+v\right\},\ \{u,\ 0,\ 1\},\ \{v,\ 0,\ 1\}\right]$$

Applying a transformation to a rectangle or a parallelepiped shows how the transformation operates. The construct also applies to atlases and many-valued manifolds (the one in the example implementing a fractal variety).

```
In[57]:= Draw[
          Mapping[deform]@Nest[Mapping[IFS["RectangularSierpinski2D"]],
            Manifold[{u, (1 - u) v}, {u, 0, 1}, {v, 0, 1}], 5],
          PlotPoints → 2, Mesh → None, ColorFunction → (ColorData[1][1] &),
          BoundaryStyle → None]
```

Out[57]=



This general scheme is supplemented by a set of common transformations in view of applications to geometry, shape design, CAD, or engineering.

A Collection of Transformations

In order to facilitate a later symbolic treatment of transformations (composition and other operations), we focus on transformations as typed objects that can be combined according to algebraic or heuristic rules, that is, an object approach to transformations. So there are at least three levels: typed symbolic objects implementing transformations, their definitions as vector functions, plus their associated matrices in the case of affine transformations. Once they are defined as vector functions, the transformations also apply to manifolds thanks to the scheme `Mapping[`*transformation*`][`*manifold*`]` or `Mapping[`*transformation*`]@`*manifold*.

Since transformations are implemented as typed objects, for example, `Translation[`v`]` with v a list, rewrite rules can be given for various combinations.

```
In[58]:= Composition[Translation[{τ, 0, 0}], Translation[{0, 1, τ}]]

Out[58]= Translation[{τ, 1, τ}]
```

Then in view of their use as vector functions, transformations have an associated subvalue, according to the informal scheme `SomeTransform[`*params*`_][`*v*`_]:=` *expression*. A complementary command defines the associated matrix in the case of affine transformations. For each affine transformation, there is a linear case and a strict affine case which is thought of as the linear version applied "about" some point. For instance, here is the case of a similarity, where *k* is either a scalar or a list (stretching).

```
Similarity[k_,pt_:0][p_List]:= k*(p-pt)+pt
```

```
In[59]:= Mapping[Similarity[{a, b}, {-1, 1}]]@
          Manifold[{u, v}, {u, 0, 1}, {v, 0, 1}]

Out[59]= Manifold[{-1 + a (1 + u), 1 + b (-1 + v)}, {u, 0, 1}, {v, 0, 1}]
```

In the three-dimensional case, there are different ways to specify a rotation: Euler angles, nautical angles, and axis and angle (or matrix exponential). These rotations can themselves be decomposed into precession, pitching, and spinning. That is why we introduce typed entities, so `Precession[`φ`]` denotes a precession of angle φ or `Euler[`φ`, `θ`, `ψ`]` denotes a rotation with the specified Euler angles. The type `Rotation` is restricted to the two-dimensional case.

The spinning direction can be given either by a vector or by two spherical angles. Precession and winding are one and the same thing; so are nutation and rolling. Here is the rotation matrix associated with the given spinning parameters, followed by the matrix of an embedding.

```
In[60]:= Row[Map[MatrixForm@Matrix@#&, {Spinning[{0, 1, 0}, α],
           Embedding[{{0, 1, 1}, {1, 1, 0}}]}], Spacer[50]]

Out[60]=  ⎛ Cos[α]  0  Sin[α] ⎞   ⎛ 0  1 ⎞
          ⎜   0     1    0   ⎟   ⎜ 1  1 ⎟
          ⎝-Sin[α]  0  Cos[α] ⎠   ⎝ 1  0 ⎠
```

These additional definitions supersede and extend those formerly found in the standard package `Geometry`Rotations``. Projections, shadows, and reflections are defined in a similar way. Common affine transformations are defined in the context `Morphology`Transformations``, common transformations associated

with coordinate systems are defined in `Morphology`Coordinates``, and transformations defining classical curves or surfaces are defined in `Morphology`Shapes``.

```
In[61]:= Mapping[Scaling[{#&, Log[10, #]&}]]@
          Manifold[{u, v}, {u, 0, 1}, {v, 1, 10^3}]
```

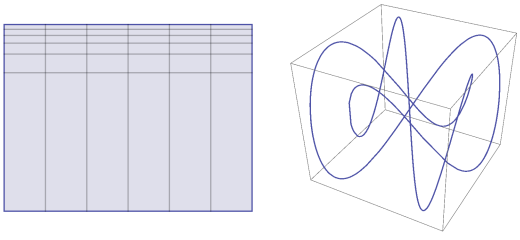
$$\text{Out[61]= Manifold}\left[\left\{u, \frac{\text{Log}[v]}{\text{Log}[10]}\right\}, \{u, 0, 1\}, \{v, 1, 1000\}\right]$$

```
In[62]:= Mapping[Lissajous[{2, 3, 5}]]@Manifold[{t}, {t, 0, 2 π}]
```

$$\text{Out[62]= Manifold}[\{\text{Sin}[2\,t], \text{Sin}[3\,t], \text{Sin}[5\,t]\}, \{t, 0, 2\,\pi\}]$$

```
In[63]:= Draw[{%%, %}, Frame → False, Axes → None, PlotPoints → {3, Automatic},
          Mesh → {5, None}, AspectRatio → {0.75, 1}]
```

Out[63]=



Applications to Manifolds and Shapes

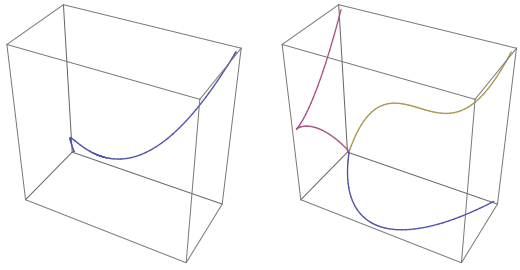
Applying various transformations to standard manifolds constitutes a way to generate all kinds of shapes by deformation. In particular, `Flattening` (projection) is more or less equivalent to `Project` from the former package `Graphics`Graphics3D``, defined in the case of three-dimensional graphics and projections onto coordinate planes. Here is a triple mapping that concisely expresses these three projections.

```
In[64]:= lifting = Lift[Manifold[{t^2, t^3}, {t, -1, 1}]]
```

$$\text{Out[64]= Manifold}\left[\left\{t, t^2, t^3\right\}, \{t, -1, 1\}\right]$$

```
In[65]:= Draw[{lifting, Mapping[{
           Flattening[{{1, 0, 0}, {0, 1, 0}}, {0, 0, 1}, {0, 0, -1}],
           Flattening[{{0, 1, 0}, {0, 0, 1}}, {1, 0, 0}, {-1, 0, 0}],
           Flattening[{{0, 0, 1}, {1, 0, 0}}, {0, 1, 0}, {0, 1, 0}]
          }]@lifting}, Axes → None]
```

Out[65]=



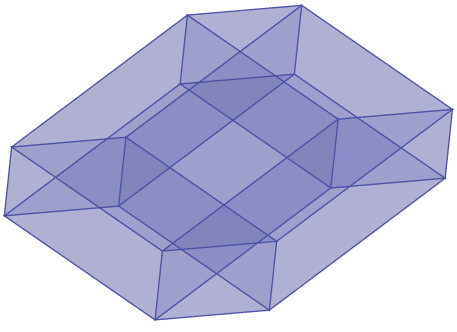
Applying a nonlinear transformation to a rectilinear object generally yields a curved object. This would not be directly possible with graphics primitives. With their analytical representations, manifolds constitute the natural primitives for nonlinear operations, while built-in graphics primitives are more or less restricted to affine operations.

Applications to Higher-Dimensional Manifolds

There is an interesting use of transformations for the two-dimensional or perspective 3D visualization of higher-dimensional manifolds. As an example, here is a two-dimensional shadow of the sides of a four-dimensional hypercube.

```
In[66]:= Draw[Mapping[Shadowing[{{1.2, 1, 0.7, 0}, {0, -1.2, 1, 0.7}},
           {{1, 0, -0.7, 1.2}, {-1, 0.7, 0, 1.2}}]]@
         Sides[UnitHyperCube[{a, b, c, d}], 2], PlotPoints → 2,
        Mesh → None, Axes → None, Frame → False, AspectRatio → 0.7]
```

Out[66]=



One can also compute a three-dimensional projection that is then displayed with the *Mathematica* internal projection engine (see the section Higher-Dimensional Manifolds).

Extrusions

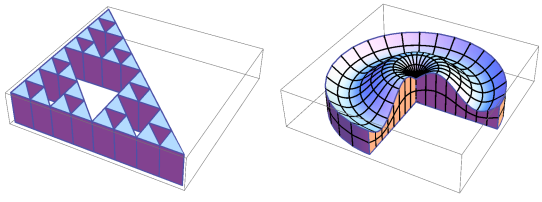
Extrusion is a feature commonly encountered in drawing or CAD applications. The analytical approach to manifolds lends itself well to extending the common extrusion process by means of parametrized transformations. This is one of the major applications of parametrized mappings, which reveals a relationship between the theoretical notion of a continuous set of transformations and the technical idea of extrusion.

```
In[67]:= Mapping[someTransformation, someParameters] @
          Manifold[{parametric}, someDomain]

Out[67]= Manifold[someTransformation[{parametric}],
          someDomain, someParameters]
```

The ordinary extrusion is no more and no less than a single parameter mapping in the case of a translation. An embedding may be necessary to initiate the process, in which case the undocumented function `Compose` conveniently expresses the composition of mappings. Many-valued manifolds as well as atlases can also be extruded.

```
In[68]:= Draw[{
          Compose[Mapping[Translation[{0, 0, h}], {h, 0, 1/5}],
           Mapping[Embedding[{{1, 0, 0}, {0, 1, 0}}]],
           Nest[Mapping[IFS["RectangularSierpinski2D"]],
            Manifold[{u, (1 - u) v}, {u, 0, 1}, {v, 0, 1}], 3]],
          Compose[Mapping[Spinning[{0, 0, 1}, θ], {θ, 0, 3 π/2}],
           Manifold[{x, 0, z (2 + Sin[x])}, {x, 0, 2 π}, {z, 0, 1}]]},
         PlotPoints → {2, {10, 3, 25}}, Mesh → {0, {7, 1, 20}},
         Axes → None, BoundaryStyle → Automatic]
```

Out[68]=



Ruled surfaces, cylindrical shapes, objects of revolution, conic objects, and others are obtained by specific extrusions. Rotational extrusions more or less supersede `RevolutionPlot3D` (formerly `SurfaceOfRevolution`), especially when used with the spinning transformation, whose first parameter corresponds to the revolution axis, and they naturally extend to solids of revolution.

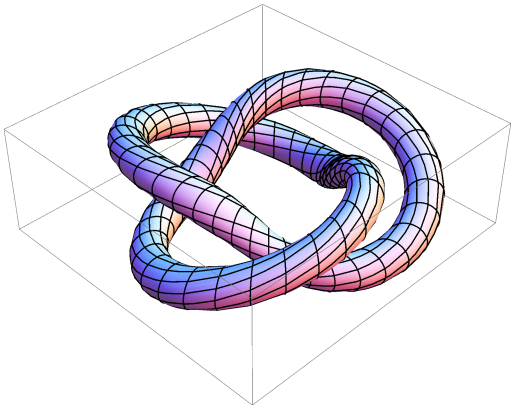
As a final example, the function `TwistedTube` introduced by Edwards in [26] is rewritten here in terms of manifolds by means of an extrusion combining a rotation, a translation, an embedding, and a spinning transformation with vertical axis.

```
TwistedTube[m:Manifold[{_,_}, ___], deltaTheta:{theta_, min_:0, max_:(2
Pi)}, r_, twist_]:=
```

```
 Mapping[Composition[
    Spinning[{0,0,1}, theta], Embedding[{{1,0,0}, {0,0,1}}],
Translation[{r,0}], Rotation[twist theta]],
 deltaTheta][m]
```

```
In[69]:= theTube = TwistedTube[Manifold[
           {Cos[u]/2, 1 + Sin[u]/2}, {u, 0, 2 Pi}], {θ, 0, 4 π}, 3, 3/2];

In[70]:= Draw[theTube, Mesh → {10, 70},
          ViewPoint → {1.9, -1.9, 2.0}, Axes → None]
```

Out[70]=

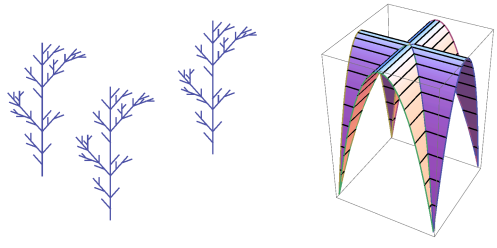


Duplications

A duplication is an accumulation of transformed manifolds that can also be viewed as the effect of a multiple mapping. By default, many-valued manifolds are generated. The IFS branch was defined earlier.

```
In[71]:= Draw[{
           Mapping[Table[Translation[{τ, Abs[τ]}], {τ, {-1/3, 0, 1/2}}]]@
            Nest[Mapping[Array[branch, 4]],
              Manifold[{0., 1. t}, {t, 0, 1}], 3],
           Mapping[Table[Precession[θ], {θ, 0, 3 π/2, π/2}]]@Manifold[
             {(1 - u) Abs[v] + u, v, 3 (1 - v^2)}, {u, 0, 1}, {v, -1, 1}]},
          PlotPoints → {2, {2, 21}}, Mesh → {0, {0, 20}}, Axes → None,
          BoundaryStyle → {None, Automatic},
          ColorFunction → {(ColorData[1][1] &), Automatic},
          AspectRatio → {0.8, Automatic}]
```

Out[71]=



Possible applications to the geometry of vaults or more general problems in architectural design are mentioned by Cerny in [27]; see also [28].

□ Right Compositions

Right compositions and their variants implement embeddings, connections, changes of coordinates, and reshapings. Here, we call a connection of two manifolds the identification of the coordinate system of the former with the codomain of the latter. In most cases, this determines a connection on the embedded manifold as soon as a connection is given for the carrier manifold. A connection can be used to define a manifold (as a figure) on some other carrier manifold. Two manifolds can be connected provided the domain of the former and the codomain of the latter have the same dimension. If the latter has a lower dimension, it can nonetheless be connected to some submanifold of the former, in which case the connection is rather an embedding. So connections and embeddings are two variants of the single operation that we call `Embed`, used in the form `Embed[`*carrier*`,` *embedded*`]` or the variant `Embed[`*carrier*`,` *mapping*`,` *embedded*`]`, where mapping is typically an embedding.

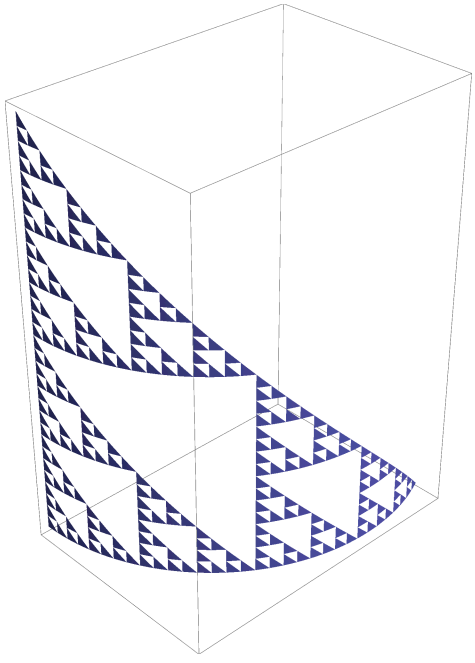
Connections

When the domain of the first manifold has the same dimension as the codomain of the second, the embedding boils down to a connection. Thanks to the connection of a many-valued manifold, a Sierpinski triangle can be pasted onto a cylinder.

```
In[72]:= cylindricalSierpinski =
           Embed[Manifold[Cylindrical[{0.5, 2 θ - π/2, h}], θ, h],
            Nest[Mapping[IFS["RectangularSierpinski2D"]],
             Manifold[{u, (1 - u) v}, {u, 0, 1}, {v, 0, 1}], 5]];
```

```
In[73]:= Draw[cylindricalSierpinski, Axes → None, PlotPoints → 2,
          Mesh → None, ColorFunction → (ColorData[1][1] &),
          BoundaryStyle → None, ViewPoint → {2., -1.9, 1.7}]
```

Out[73]=



Changes of Coordinates and Reshapings

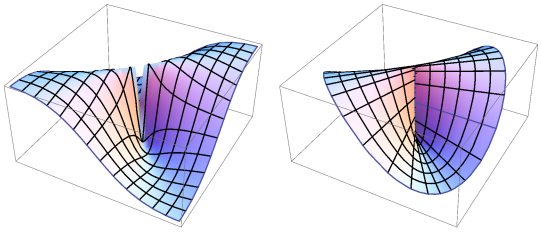
When the dimensions are the same, an embedding is no more and no less than a change of coordinates associated with a change of boundary, so it can also be viewed as the reshaping of the second argument or else as its clipping on the first one. Goetz and Wagon used such changes of coordinates in [29] as a means to carry out adaptive surface plotting. The second example is adapted from Kuzniarek [30].

```
In[74]:= m1 = Manifold[{x, y, x y/(x^2 + y^2)}, {x, -1, 1}, {y, -1, 1}];
         m2 = MapAt[Simplify, Embed[m1,
             Manifold[{r Cos[θ], r Sin[θ]}, {r, 0, 1}, {θ, 0, 2 π}]], 1]
```

```
Out[75]= Manifold[{r Cos[θ], r Sin[θ], Cos[θ] Sin[θ]}, {r, 0, 1}, {θ, 0, 2 π}]
```

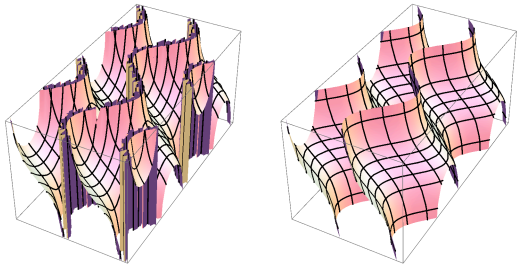
```
In[76]:= Draw[{m1, MapAt[# /. θ → θ + Sin[4 θ] / 10 &, m2, 1]}, Axes → None,
          BoundaryStyle → Automatic, Mesh → {{12, 12}, {5, 30}}]
```

Out[76]=



```
In[77]:= m1 = Manifold[{x, y, Cot[x + Sin[y]]}, {x, -π, π}, {y, -2 π, 2 π}];
         m2 = Embed[m1, Manifold[{u - Sin[v], v}, {u, -π, π}, {v, -2 π, 2 π}]]
Out[77]= Manifold[{u - Sin[v], v, Cot[u]}, {u, -π, π}, {v, -2 π, 2 π}]

In[78]:= Draw[{m1, m2}, Axes → None,
          PlotRange → {{{-π, π}, {-2 π, 2 π}, {-3, 3}}}]
```

Out[78]=



Strict Embeddings

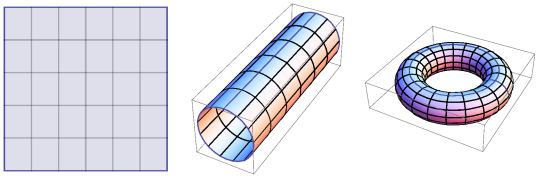
We sometimes need to embed a manifold, a many-valued manifold, or an atlas into a higher-dimensional manifold. Such a strict embedding is determined by a point where the origin of the embedded manifold is posted and the specification of its orientation (typically with the `Embedding` transformation) or more generally by a submanifold that carries the embedded one.

```
In[79]:= Embed[Manifold[Polar[{ρ, θ}], ρ, θ],
          Mapping[Embedding[{1, 2}, {1, 0}]], Manifold[{t}, {t, 0, Pi}]]
Out[79]= Manifold[{(1 + t) Cos[2 t], (1 + t) Sin[2 t]}, {t, 0, π}]

In[80]:= Embed[Manifold[Polar[{ρ, θ}], ρ, θ],
            Manifold[{1, 2} u + {1, 0}, u], Manifold[{t}, {t, 0, Pi}]] == %
Out[80]= True
```

The embedding constitutes a powerful shape generator. In the following example, a torus is built by two successive embeddings: a square is first embedded into a cylindrical coordinate system with a convenient axis, which yields a cylinder that in turn, is embedded into another cylindrical coordinate system with an orthogonal axis, which yields the torus.

```
In[81]:= t0 = Manifold[{u, v}, {u, 0, 2 π}, {v, 0, 2 π}];
         t1 = Mapping[Permutation[{2, 3, 1}]][
            Embed[Manifold[Cylindrical[{1, θ, z}], θ, z], t0]];
         t2 = Embed[Manifold[Cylindrical[{r + 3, θ, z}], r, θ, z], t1];

In[84]:= Draw[{t0, t1, t2}, Frame → False, Mesh → {{5, 4}, {12, 5}, {12, 20}},
          BoundaryStyle → Automatic, Axes → None]
```

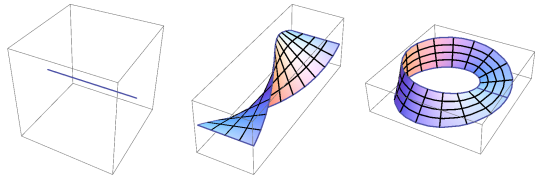
Out[84]=



Similarly, a Moebius strip is built by embedding a helicoid into a cylindrical coordinate system. The helicoid is obtained by extruding a segment by a left screw.

```
In[85]:= m0 = Manifold[{t, 0, 0}, {t, -1, 1}];
         m1 = Mapping[LeftScrewing[{0, 2, 0}, λ], {λ, 0, π}] @m0

Out[86]= Manifold[{t Cos[λ], 2 λ, t Sin[λ]}, {t, -1, 1}, {λ, 0, π}]

In[87]:= m2 = Embed[Manifold[Cylindrical[{r + 3, θ, z}], r, θ, z], m1];
         Draw[{m0, m1, m2}, Mesh → {None, {3, 12}, {3, 15}},
          BoundaryStyle → Automatic, Axes → None]
```

Out[87]=



An unusual form of the Klein bottle [9, page 239] is similarly built by embedding the twisted extrusion of a figure eight into a cylindrical coordinate system or by applying `TwistedTube` to the initial figure eight.

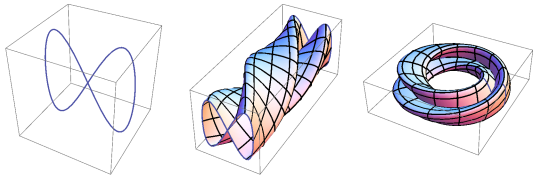
```
In[88]:= k0 = Manifold[{Sin[t], 0, Sin[2 t]}, {t, 0, 2 π}];
         k1 = Mapping[LeftScrewing[{0, 2, 0}, λ/2], {λ, 0, 2 π}] @k0
```

Out[89]=

$$\text{Manifold}\left[\left\{\text{Cos}\left[\frac{\lambda}{2}\right]\text{Sin}[t]-\text{Sin}[2t]\,\text{Sin}\left[\frac{\lambda}{2}\right],\ \lambda,\ \text{Cos}\left[\frac{\lambda}{2}\right]\text{Sin}[2t]+\text{Sin}[t]\,\text{Sin}\left[\frac{\lambda}{2}\right]\right\},\ \{t,0,2\pi\},\ \{\lambda,0,2\pi\}\right]$$

```
In[90]:= k2 = Embed[Manifold[Cylindrical[{3 + r, θ, z}], r, θ, z], k1];
         k2 == TwistedTube[
           Manifold[{Sin[t], Sin[2 t]}, {t, 0, 2 π}], {λ, 0, 2 π}, 3, 1/2]
Out[91]= True

In[92]:= Draw[{k0, k1, k2}, Mesh → {None, {15, 10}, {15, 12}},
          BoundaryStyle → Automatic, Axes → None]
```

Out[92]=



In all cases, an appropriate mapping, derived from the carrier manifold, yields the same result as the embedding.

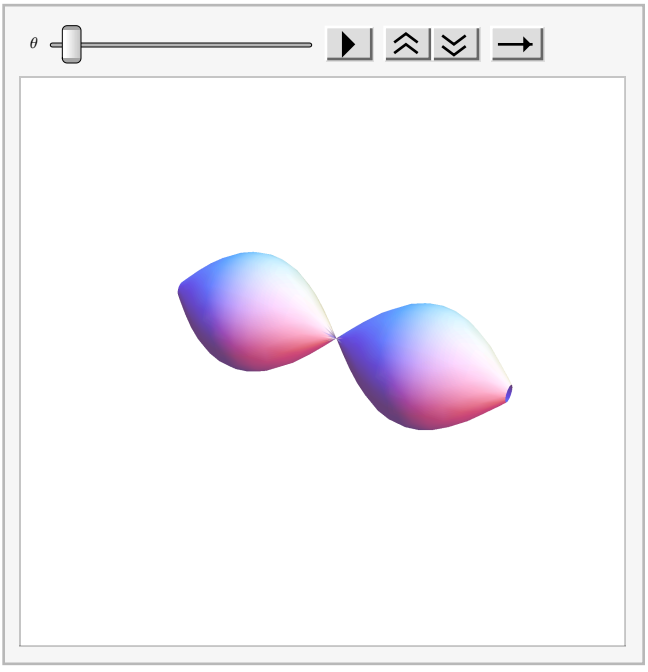
```
In[93]:= {t2, m2, k2} ==
          Map[Mapping[Cylindrical[# + {3, 0, 0}] &], {t1, m1, k1}] ==
          Map[Embed[Manifold[Cylindrical[{r, θ, z}], r, θ, z],
             Mapping[Translation[{3, 0, 0}]], #] &, {t1, m1, k1}]
Out[93]= True
```

□ Animations and Ray Tracing

Since Version 6, `Animate` and `Manipulate` supersede the various movie-plotting commands of the package `Graphics`Animation``. The first example shows an ordinary motion, while the second one visualizes an extrusion. Load the package `Morphology` and evaluate the Inputs to activate them.

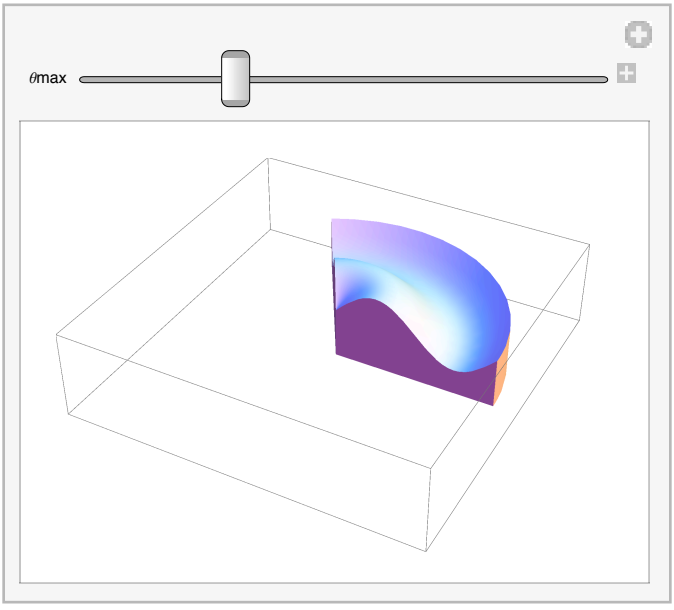
In[94]:=
```
Animate[Draw[Mapping[Precession[θ]]@Manifold[
    {u, Cos[v] Sin[u], Sin[v] Sin[u]}, {u, -Pi, Pi}, {v, 0, 2 Pi}],
  Axes → None, Mesh → None, Boxed → False,
  PlotRange → {{-3, 3}, {-3, 3}, {-2, 2}}],
 {θ, 0, π}, AnimationRunning → False]
```

Out[94]=



In[95]:=
```
Manipulate[
 Draw[Compose[Mapping[Spinning[{0, 0, 1}, θ], {θ, 0, θmax}],
   Manifold[{x, 0, z (2 + Sin[x])}, {x, 0, 2 π}, {z, 0, 1}]],
  Mesh → None, Axes → None, PlotRange →
   {{-7, 7}, {-7, 7}, {-0.5, 3}}], {{θmax, 2 π / 3}, 0.01, 2 π}]
```

Out[95]=



Finally, a tool developed by Maeder to convert and export surface graphics data to a ray tracing program deserves to be mentioned [31], a feature from now on replaced by `Export`.

□ Homotopy (Morphing) and Interpolation

Provided that linear operations can be defined with respect to some type of objects, the homotopic transformation $(1-k)\,O_1 + k\,O_2$, $k \in [0, 1]$ defines all intermediaries between the objects O_1 and O_2, that is, a linear interpolation between these two objects. In the case of manifolds, the idea can be generalized to nonlinear interpolation, provided weighting functions are specified. It can also be generalized to polynomial interpolation.

In[96]:=
```
{Homotopy[k][g1[x], g2[x]],
 Homotopy[{k, k0, k1}][g1[x], g2[x]], Homotopy[w[k]][g1[x], g2[x]]}
```

Out[96]=
$$\left\{(1-k)\,g1[x] + k\,g2[x],\ \frac{g2[x]\,(k-k_0) - g1[x]\,(k-k_1)}{-k_0+k_1},\ g1[x]\,(1-w[k]) + g2[x]\,w[k]\right\}$$

```
In[97]:= Mapping[Homotopy[k], {k, 0, 1}][
          Manifold[{1, t, t^2}, {t, 0, 1}],
          Manifold[{t, t^2, t^3}, {t, 0, 1}]
         ]

Out[97]= Manifold[{1 - k + k t, (1 - k) t + k t^2, (1 - k) t^2 + k t^3},
          {t, 0, 1}, {k, 0, 1}]
```

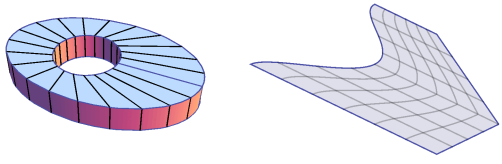
Homotopy is a powerful shape generator that produces n-dimensional manifolds by generating all the intermediaries between two $(n-1)$-dimensional manifolds (currently valid only for single-valued manifolds). If the domains are not identical, the manifolds must first be normalized. Homotopy can be used, for instance, to design shapes with a hole.

In some cases, geometric domains are presented in the form of the so-called cylindrical decompositions: $u_{\min} \le u \le u_{\max}$, $v_{\min}[u] \le v \le v_{\max}[u]$, $w_{\min}[u, v] \le w \le w_{\max}[u, v]$… . That is typically the way integration domains are specified. Homotopy then transforms those patches into manifolds. This idea was suggested by Tavouksoglou and Freed in [32, 33].

```
In[98]:= Patch[{x, y, z}, {x, 0, 1}, {y, 0, 1 - x}, {z, 0, 1 - x - y}]

Out[98]= Manifold[{x, (1 - x) y, (1 - x - (1 - x) y) z},
          {x, 0, 1}, {y, 0, 1}, {z, 0, 1}]
```

```
In[99]:= Draw[{
           Mapping[Homotopy[k], {k, 0, 1}][
            Manifold[{Cos[θ] - 1/2, Sin[θ], h}, {θ, 0, 2 π}, {h, 0, 1}],
            Manifold[{3 Cos[θ], 2 Sin[θ], h}, {θ, 0, 2 π}, {h, 0, 1}]
           ],
           Patch[{u + v, u - v}, {u, -1, 1},
            {v, u^4 - 2 u^2, 2 - (u + Abs[u])}]}, Mesh → {{20, 0, 0}, {7, 5}},
          Axes → None, Boxed → False, ViewPoint → {{0.80, -1.70, 1.8}},
          Frame → False, BoundaryStyle → Automatic, AspectRatio → 0.6]
```

Out[99]=



■ Future Directions

Thanks to their analytical potential, manifolds constitute a natural foundation for differential geometry, field theory, and also some modeling applications. A data structure is introduced to describe fields over manifolds. The principle of an extension to manifolds of field analysis is described. Finally, a link with a finite element package is presented.

□ Fields

Let us represent fields by typed entities, the arguments of which are a scalar or a rectangular array (for tensors), a list of variance specifications (except in the scalar case), and the manifold over which the field is defined. This object approach to fields stresses the indissoluble link between the field components and the manifold coordinates. `Normal` extracts the scalar or the array.

```
In[100]:= m = Manifold[Polar[{ρ, θ}], {ρ, 0, 1}, {θ, 0, 2 π}];
          potential = Field[φ[ρ, θ], m]

Out[101]= Field[φ[ρ, θ], Manifold[{ρ Cos[θ], ρ Sin[θ]}, {ρ, 0, 1}, {θ, 0, 2 π}]]

In[102]:= Normal[potential]

Out[102]= φ[ρ, θ]
```

Some fields, like the metric field or the field of Christoffel coefficients, derive from the manifold itself, so they are expressed as functions of the manifold.

Christoffel Coefficients

When the codomain is a Euclidean n-dimensional space with its canonical connection, which is the default assumption, an induced connection is determined on the manifold. Although they are not tensorial, Christoffel coefficients are usually manipulated like tensors.

```
In[103]:= ChristoffelGamma[{High, Low, Low}][m] // Normal

Out[103]= {{{0, 0}, {0, -ρ}}, {{0, 1/ρ}, {1/ρ, 0}}}
```

The Christoffel coefficients are then used to compute covariant derivatives.

Vector and Tensor Analysis

Vector and tensor analysis on manifolds are based on the covariant derivative `CovariantD`; fields are represented as specified earlier. For antisymmetric tensors, the wedge product and the exterior derivative (`ExteriorD`) are also introduced.

```
In[104]:= CovariantD[Field[φ[u, v], Manifold[{u, v}, u, v]]] // Normal

Out[104]= {φ^(1,0)[u, v], φ^(0,1)[u, v]}

In[105]:= CovariantD[Field[ρ Cos[θ], m]] // AbridgedForm

Out[105]= Field[{Cos[θ], -ρ Sin[θ]}, {Low}, -Manifold-]
```

□ Differential Geometry

The basic ingredients of differential geometry are the tangent manifold (Jacobian computation) and the metric. The metric, the Christoffel coefficients, and other differential characteristics (e.g., curvature or torsion) are common to differential geometry and field analysis.

Tangent Manifold

One must distinguish the tangent manifold computed at some point, which is a plane or a hyperplane, from the field of tangent manifolds that can be viewed as a manifold over an abstract space with twice the initial dimension (the tangent bundle).

```
In[106]:= m = Manifold[{x^2 + y^2, -2 x y}, x, y];

In[107]:= TangentManifold[m, {-1, 1}]

Out[107]= Manifold[{-2 Dt[x] + 2 Dt[y], -2 Dt[x] + 2 Dt[y]}, Dt[x], Dt[y]]

In[108]:= TangentBundle[m]

Out[108]= Manifold[{2 x Dt[x] + 2 y Dt[y], -2 y Dt[x] - 2 x Dt[y]},
           {x, y, Dt[x], Dt[y]}]
```

Metric

When the codomain is a Euclidean n-dimensional space with its canonical metric, which is the default assumption, an induced metric is determined on the manifold.

```
In[109]:= m = Manifold[Annular[R][{r, θ, φ}], {θ, 0, π}, {φ, 0, 2 π}];
          Metric[{Low, Low}][m] // Normal // MatrixForm
```

$$\text{Out[110]//MatrixForm=}\quad \begin{pmatrix} (R + r\,\text{Cos}[\varphi])^2 & 0 \\ 0 & r^2 \end{pmatrix}$$

The associated contravariant tensor is no more and no less than its inverse.

```
In[111]:= Normal[Metric[{Low, Low}][m]].
            Normal[Metric[{High, High}][m]] // MatrixForm
```

$$\text{Out[111]//MatrixForm=}\quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

When the metric structure of the codomain is not Euclidean, an explicit metric can be given; for example, the Poincaré half-plane metric.

```
In[112]:= PoincareMetric[λ][{u, v}] // MatrixForm
```

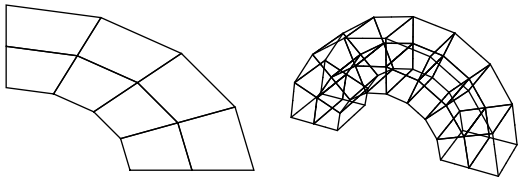
$$\text{Out[112]//MatrixForm=}\quad \begin{pmatrix} \frac{\lambda^2}{v^2} & 0 \\ 0 & \frac{\lambda^2}{v^2} \end{pmatrix}$$

□ Applications to Mechanical Engineering

Beyond obvious applications to space or spacetime modeling, manifolds have uses in mechanical engineering, especially by means of interactions with finite element packages. The details depend on the data structure used for finite elements. Here is an example with the "IMTEK *Mathematica* Supplement" package [34], assumed to have been installed [35] (the packages are loaded by `Mesher`); see also [36] and [37].

```
In[113]:= Needs["Morphology`Mesher`"];
          m2D = Manifold[{3 r Cos[t] / 2, r Sin[t]}, {r, 1, 2}, {t, 0, π / 2}];
          nexus2D = ToImsNexus[m2D, {r, 3}, {t, 5}];
          m3D =
           Manifold[{r Cos[t], h, r Sin[t]}, {r, 1, 2}, {h, 0, 1}, {t, 0, π}];
          nexus3D = ToImsNexus[m3D, {r, 3}, {h, 2}, {t, 9}]

Out[115]= - imsNexus -

In[116]:= GraphicsRow[{
            Graphics[ imsDrawElements[ nexus2D ] , AspectRatio → Automatic] ,
            Graphics3D[ imsDrawElements[ nexus3D ] ,
             Boxed → False, ViewPoint → {1., -2., 1.2}]
           }]
```

Out[116]=



`nexus2D` or `nexus3D` are not only meshes but structures differentiating boundary nodes from interior nodes, able to take into account boundary values in view of further computations.

■ Discussion and Prospects

Founded on a reification of parametric representations, this computational approach to manifolds leads to a uniform treatment of questions that arise in differential geometry and field theory plus other domains such as shape design, fractal generation, scene description, or mesh generation. In particular, it has the potential to supersede various graphics commands. Numerous operators generate more or less intricate or distorted manifolds by twisting and combining simpler ones.

As opposed to the functional approach of plotting commands that build shapes by assembling low-level graphics primitives, manifolds lead to a higher-level description of shapes that can be defined by compact symbolic expressions, rather than huge assemblies of raw graphics primitives. This symbolic layer paves the way for a concise representation of many-level systems, then processed as wholes, such as spatial scenes, linkages, or other assembled systems.

Nevertheless, the set of packages developed for that purpose is primarily intended for investigating the feasibility of this project: testing has remained minimal, there are no messages, exceptions have not been investigated thoroughly, and the packages might have to be reorganized. The system of options associated with manifolds, atlases, and the `Draw` command should be improved with a better filtering mechanism. The option specification `DrawingStyle` is an experimental feature that could be discarded.

In the case of intricate systems, a mechanism for naming and retrieving the subexpressions describing the corresponding subsystems would be welcome. For instance, naming individual figures would occasionally be useful in the case of geometric scenes. Also, it would be interesting to enable links with geometry packages like *Geometrica*05 [38].

Parametric representations do not lend themselves well to the algebraic approach, from which a substantial part of the power of computer algebra derives, which weakens the idea of a full symbolic treatment of form. Because implicitization and parametrization of manifolds remain unsolved problems (except in particular cases) [39], the relationship between parametric representations and implicit definitions (Cartesian descriptions or inequalities) remains loose. Consequently, the analytical treatment of manifolds does not lend itself well to Boolean operations, which require algebraic computations.

The affine transformations defined in the context `Morphology`Transformations`` have been introduced mainly in view of applications to mechanical engineering. Their compatibility with the novel set of geometric transformations of Version 6 should undergo further investigation.

Nevertheless, the major role of the analytical approach in physics gives importance to this way of representing and manipulating manifolds; so does its capability to blend and supplement a variety of tools, scattered about the kernel and various packages.

■ Conclusion

Founded on reified parametrizations, the algorithmic approach to manifolds presented here leads to a generic treatment of form modeling that encompasses shape design, differential geometry, and field analysis, with direct applications to mechanical engineering. It introduces a unified viewpoint that not only gathers and supplements a variety of graphics tools scattered about the kernel and several packages, but also enables a symbolic approach to form that concisely encodes the various entities encountered in form modeling. As such, it constitutes a possible foundation for a computational morphology.

■ References

[1] X-S. Gao and D. Wang, eds., *Mathematics Mechanization and Applications*, New York: Academic Press, 2000.

[2] G. Lakoff and R. Nunez, *Where Mathematics Come From: How the Embodied Mind Brings Mathematics into Being*, New York: Basic Books, 2000.

[3] E. A. Lord and C. B. Wilson, *The Mathematical Description of Shape and Form*, New York: Halsted Press, 1986.

[4] R. Barrère, "The Structuring Power of *Mathematica* in Mathematics and Mathematical Education," (Paper #35) in *Electronic Proceedings of the Third International Mathematica Symposium (IMS'99)*, Hagenberg, Austria (V. Keränen, ed.), 1999. south.rotol.ramk.fi/keranen/IMS99/ims99papers/ims99papers.html. library.wolfram.com/infocenter/Conferences/6138.

[5] T. Wickham-Jones, *Mathematica Graphics: Techniques & Applications*, New York: Springer-Verlag, 1994.

[6] M. Trott, *The Mathematica GuideBook for Graphics*, New York: Springer-Verlag, 2004.

[7] M. Berger and B. Gostiaux, *Differential Geometry: Manifolds, Curves and Surfaces*, Graduate Texts in Mathematics 115, New York: Springer-Verlag, 1992.

[8] J. Oprea, Differential Geometry and Its Applications, 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 2004.

[9] A. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica*, 2nd ed., Boca Raton, FL: CRC Press, 1997. library.wolfram.com/infocenter/Books/3759.

[10] Y. Tazawa, "Experiments in the Theory of Surfaces," (Paper #37) in *Electronic Proceedings of the Third International Mathematica Symposium (IMS'99)*, Hagenberg, Austria (V. Keränen, ed.), 1999. south.rotol.ramk.fi/keranen/IMS99/ims99papers/ims99papers.html. library.wolfram.com/infocenter/Conferences/6177.

[11] Y. Tazawa, "Gauss-Bonnet Theorem by *Mathematica*," in *Symbolic Computation: New Horizons, Proceedings of the Fourth International Mathematica Symposium (IMS'01)*, Tokyo, Japan (Y. Tazawa, ed.), Tokyo: Tokyo Denki University, 2001 pp. 485-492. library.wolfram.com/infocenter/Books/3598.

[12] J. Hoschek, D. Lasser, and L. L. Schumaker, *Fundamentals of Computer Aided Geometric Design*, Wellesley, MA: A K Peters, Ltd., 1993.

[13] B. Chazelle, "Computational Geometry: A Retrospective," in *Proceedings ot the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC'94)*, Montreal, Canada, New York: Association for Computing Machinery, 1994 pp. 75-94. DOI-Link: doi.acm.org/10.1145/195058.195110.

[14] L. Piegl and W. Tiller, *The NURBS Book*, New York: Springer-Verlag, 1997.

[15] C. M. Grimm and J. F. Hughes, "Modeling Surfaces of Arbitrary Topology Using Manifolds," in *Proceedings of the Twenty-Second Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'95)*, Los Angeles, CA: Association for Computing Machinery, 1995 pp. 359-368. DOI-Link: doi.acm.org/10.1145/218380.218475.

[16] J. W. Gray, *Mastering Mathematica: Programming Methods and Applications*, New York: Academic Press, 1994. library.wolfram.com/infocenter/Books/3791.

[17] R. E. Maeder, *Computer Science with Mathematica: Theory and Practice for Science, Mathematics, and Engineering*, Cambridge: Cambridge University Press, 2000.

[18] B. Bastl, "Computer Aided Geometric Design in *Mathematica*," 2005 Wolfram Technology Conference, Champaign, IL, 2005. library.wolfram.com/infocenter/Conferences/5812.

[19] A. Noll, "A Computer Technique for Displaying *n*-Dimensional Hyperobjects," *Communications of the ACM*, **10**(8), 1967 pp. 469-473.

[20] S. Kivelä, "On the Visualization of Riemann Surfaces," in *Applied Mathematica, Electronic Proceedings of the Eighth International Mathematica Symposium (IMS'06)*, Avignon, France (Y. Papegay, ed.), Sophia Antipolis, France: INRIA, 2006 ISBN 2-7261-1289-7.
internationalmathematicasymposium.org/IMS2006/IMS2006_CD/html/articles.html.

[21] D. H. von Seggern, *CRC Standard Curves and Surfaces*, Boca Raton, FL: CRC Press, 1993.

[22] M. Attéia and J. Gaches, *Approximation Hilbertienne: Splines, Ondelettes, Fractales*, Les Ulis and Grenoble: EDP Sciences and Presses Universitaires de Grenoble (PUG), 1999.

[23] M. Barnsley, *Fractals Everywhere*, San Diego, CA: Academic Press, 1988.

[24] S. Wagon, *Mathematica in Action*, 2nd ed., New York: W. H. Freeman & Company, 1991 pp. 102-108.

[25] J. M. Gutiérrez, A. Iglesias, M. A. Rodriguez, and V. J. Rodriguez, "Fractal Image Generation Using Iterated Function Systems," in *Mathematics with Vision, Proceedings of the First International Mathematica Symposium (IMS'95)*, Southampton, England (V. Keränen and P. Mitic, eds.), Southampton: Computational Mechanics Publications, 1995 pp. 175-182.
library.wolfram.com/infocenter/Articles/1533.

[26] C. H. Edwards, "Twisted Tubes," *The Mathematica Journal*, **3**(1), 1993 pp. 10-13.

[27] J. Cerny, "Geometry and Architecture," in *Proceedings of the Ninth European Society for Engineering Education (SEFI) European Seminar on Mathematics in Engineering Education*, Espoo, Finland (M. Demlové, L. Mustoe, and B. Olsson-Lehtonen, eds.) Helsinki, Finland: Arcada Polytechnic, 1998 pp. 27-30.

[28] P. Morel and M. Teissier, "Architecture, Set Design, and Mathematical Pattern," in *New Ideas in Symbolic Computation, Electronic Proceedings of the Sixth International Mathematica Symposium (IMS'04)*, Banff, Alberta, Canada (P. Mitic, C. J. Jacob, and J. Carne, eds.), Hampshire, England: Positive Corporation Limited, 2004.
library.wolfram.com/infocenter/Conferences/6057.

[29] R. Goetz and S. Wagon, "Adaptive Surface Plotting: A Beginning," *Mathematica in Education and Research*, **5**(3), 1996 pp. 74-83.

[30] A. Kuzniarek, "The Graphics Designer: Making the Perfect Picture," *The Mathematica Journal*, **4**(4), 1994 pp. 54-60.

[31] R. Maeder, "The *Mathematica* Programmer: Ray Tracing and Graphics Extensions," *The Mathematica Journal*, **4**(3), 1994 pp. 48-55.

[32] A. N. (Tom) Tavouktsoglou and B. Freed, "Parametric Representations of Surfaces over Arbitrary Domains," *Mathematica in Education and Research*, **3**(1), 1994 pp. 20-23.
library.wolfram.com/infocenter/Articles/1116.

[33] A. N. (Tom) Tavouktsoglou and B. Freed, "Drawing Mathematical Solids," *Mathematica in Education and Research*, **3**(4), 1994 pp. 22-24.

[34] O. Rübenkönig, Z. Liu, and J. Korvink, "Integrated Engineering Development Environment," *The Mathematica Journal*, **10**(3), 2007 pp. 562-578.
www.internationalmathematicasymposium.org.

[35] O. Rübenkönig and J. Korvink, *IMTEK Mathematica Supplement (IMS)*, (2002-2005). www.imtek.de/simulation/mathematica/IMSweb.

[36] N. M. Dai, "A Language to Solve Finite Element Problems," *The Mathematica Journal*, **8**(1), 2001 pp. 126-146.

[37] E. A. Malsch and G. Dasgupta, "Algebraic Construction of Smooth Interpolants on Polygonal Domains," *The Mathematica Journal*, **9**(3), 2005 pp. 641-658.

[38] B. Autin, "*Geometrica*05," 2005 Wolfram Technology Conference, Champaign, IL, 2005. library.wolfram.com/infocenter/Conferences/5846.

[39] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, 2nd ed., New York: Springer-Verlag, 1997.

R. Barrère, "An Algorithmic Approach to Manifolds," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.11.2–5.

■ Available Material

An experimental set of packages was developed to investigate the ideas presented in this article. It is available at macmaths.ens2m.fr/Mathematica/packages/Morphology.alpha3.zip (see also the Packages link at macmaths.ens2m.fr/*Mathematica*).

■ Acknowledgments

On the occasion of academic projects, students have contributed by experimenting with drafts of the material presented here and the author thanks them. A few student project reports are available at macmaths.ens2m.fr/students (Published Projects).

About the Author

Rémi Barrère received both his undergraduate (in 1980, in engineering) and doctoral (in 1985, in mathematics) degrees from the University of Franche-Comté in Besançon, France. Barrère first obtained a position at the University of Paris XIII and now teaches at his alma mater, where he developed the use of computer algebra and symbolic programming techniques for mathematical modeling. He teaches both mathematical modeling, as well as applied mathematics, utilizing an experimental method: he presents scientific computing to his students by having them develop projects using *Mathematica*. He has been using this method since the early 1990s in teaching as well as research, particularly in the domains of symbolic approximations and foundational questions. He is a member of the Wolfram Education Group and is the author of a book on *Mathematica* (*Mathematica*: *Calcul formel et programmation symbolique pour l'informatique scientifique*, Paris: Vuibert, 2002).

Rémi Barrère
University of Franche-Comté, ENSMM
26 chemin de l'Epitaphe
F-25000 Besançon, France
rbarrere@ens2m.fr
macmaths.ens2m.fr